Chapter 4

UML Approach to Markov Reliability Modeling

4.1 Introduction

A software system can be analyzed for its quality from many perspectives such as its performance, scalability, maintainability, usability and reliability. In general, software reliability is among the major factors in software quality ". . . since it quantifies software failures, which can make a powerful system inoperative" [67]. In safety critical systems like Nuclear Power Plant, Aerospace and Medical applications the reliability requirements are very high. The study of reliability as the quantification of the operational behavior of software systems with respect to user requirements is defined as SRE. The classic SRE process includes four major steps [11]: (i) reliability objective (ii) operational profile (iii) reliability modeling (iv) reliability validation. A reliability objective specifies the reliability target of the software with respect to the end user. The failures of the system can be classified into many types [68] as given in table 4.1. A Reliability objective defines the type of failures that the end user wants to measure.

Once failure classification is performed, reliability requirements can be defined using reliability metrics. The construction of an operational profile is important in order to select test cases according to the usage of the system [67]. A fault must be executed to

Table 4.1: Failure classification of system.

Failure Class	**Description**
Transient	Occurs with certain inputs
Permanent	Occurs with all inputs
Recoverable	System can recover without operator intervention
Unrecoverable	Operation intervention is needed to recover from failure
Non-corrupting	Failure does not corrupt system state or data
Corrupting	Failure corrupts system state or data

cause a failure, otherwise it is just a dormant fault [69]. Reliability modeling is essential to the reliability prediction and estimation process. Most of the reliability modeling approaches attempt to predict software reliability in the later stages of the life cycle [67], out of them, SGRM is the most popular one. Those models have been widely used to predict reliability in the later phases of software by modeling the number of faults and the failure rate as testing progresses. As a result of those testing, errors are found and removed and hence the reliability improves. If the reliability objective is not met, more testing will be applied in an iterative process. Finally in reliability validation, the projected and observed reliabilities are compared.

Though almost 200 SGRMs have been proposed, as per Michael R. Lyu [67], SRE is not yet fully delivering its promise. A major reason behind it is that SGRMs are measurement-based models and hence employed in isolation at the later stages of the SDLC. Also, no generic model can be fitted to all the kinds of software. Further, such models treat the software as a monolithic whole without concern to its internal structure.

Ambiguous, inconsistent and incomplete requirements; and defect is design; are the major factors for software failure. Ambiguous, inconsistent and incomplete requirements can also lead to a defective design. Hence there is a need to involve all the stakeholders, especially clients, in constructing a reliability model to ensure that none of the requirements have missed out. Hence there must be a common language to construct the reliability model.

We postulate it is possible to address this gap by providing a methodology to support SRE from requirements to deployment, with adequate analysis, through appropriate

mappings. This will address the modeling limitations of the current approaches, which have been stated in section 1.1. We propose an approach to predict software system reliability from scenario specifications, which involve extending a scenario specification to model (i) the probability of component failure and (ii) scenario transition probabilities derived from an operational profile of the system. We also carry out an empirical sensitivity analysis of the system reliability as a function of (i) the components' reliability and (ii) the transition probability between scenarios. We describe a scenario specification of the control logic of a running safety critical system of a Nuclear Power Plant as a case study.

4.2 A Case Study

Software systems are becoming more and more complex and hence they are developed in parts, known as components, which are responsible to perform their pre-defined functions. These components have well defined interfaces through which, they are integrated to construct the overall software system. Therefore, analyzing the software reliability requires models that include software components and how they weave together. Since the reliability of the system is the function of the reliability of every component in the system, therefore, to predict the reliability estimate of the overall system, there is a requirement to predict the reliability estimate of each component. In order to do that, we demonstrate our approach of reliability prediction on a small control logic component of a running safety critical CBS, known as ECCS, of Indian Nuclear Power Plant. We characterize what constitutes the specification of a control logic component.

4.2.1 Short description of ECCS

Nuclear reactors are used for generating electricity from nuclear fission reaction. Heat from nuclear fission is passed to a working fluid (water or gas), which runs through turbines. A nuclear coolant is circulated past the reactor core to absorb the heat that it generates. The coolant transfers the heat to the steam generators. Steam generators

generate steam from this heat, which is used to rotate the turbine to produce electricity. The coolant flows through the mechanical pipes and valves. There can be LOCA, if mechanical pipes or valves break due to any reason. In that case there is an external mechanism to cool down the reactor core to prevent the melting of the fuel. The melting of the fuel leads to the radiation exposure to the public or can also leads to explosion. The radiation can cause cancer and thyroid destruction. This mechanism of mitigating the consequences of LOCA is performed by a safety critical system, known as ECCS.

4.2.2 ECCS Instrumentation

The instrumentation of ECCS can be understood through its mimic diagram, shown in figure 4.1. It contains various mechanical equipments like tanks, pipes, valves and pumps. In case of LOCA, nitrogen tank pressurizes heavy water and light water to inject into the reactor core. The water injects in three phases: first phase is heavy water injection under high pressure, second phase is light water injection under intermediate pressure and finally in third stage, suppression pool system under low pressure comes. Heavy water is injected under high pressure because initially reactor core is highly pressurized and hence low pressure cannot inject the water into the core. The use of heavy water in the first stage ensures the integrity of the reactor core in case of any transient, instead of actual LOCA. As soon as the pressure comes down, in case of break in PHT circuit, intermediate pressure is capable to inject the light water. There is a suppression pool of higher capacity (3000 cubic meter) and the drain lines are connected from F/M vaults and F/M service area. All the heavy water and light water passes through core and gets dumped into the suppression pool. Therefore when the two light water tanks become empty, water from the suppression pool can be circulated continuously through pumps and heat exchangers. The temperature of water in the suppression pool increases because water transfers the heat of the reactor core into the suppression pool. Heat exchangers are used to maintain the water temperature in the suppression pool within permissible limit.

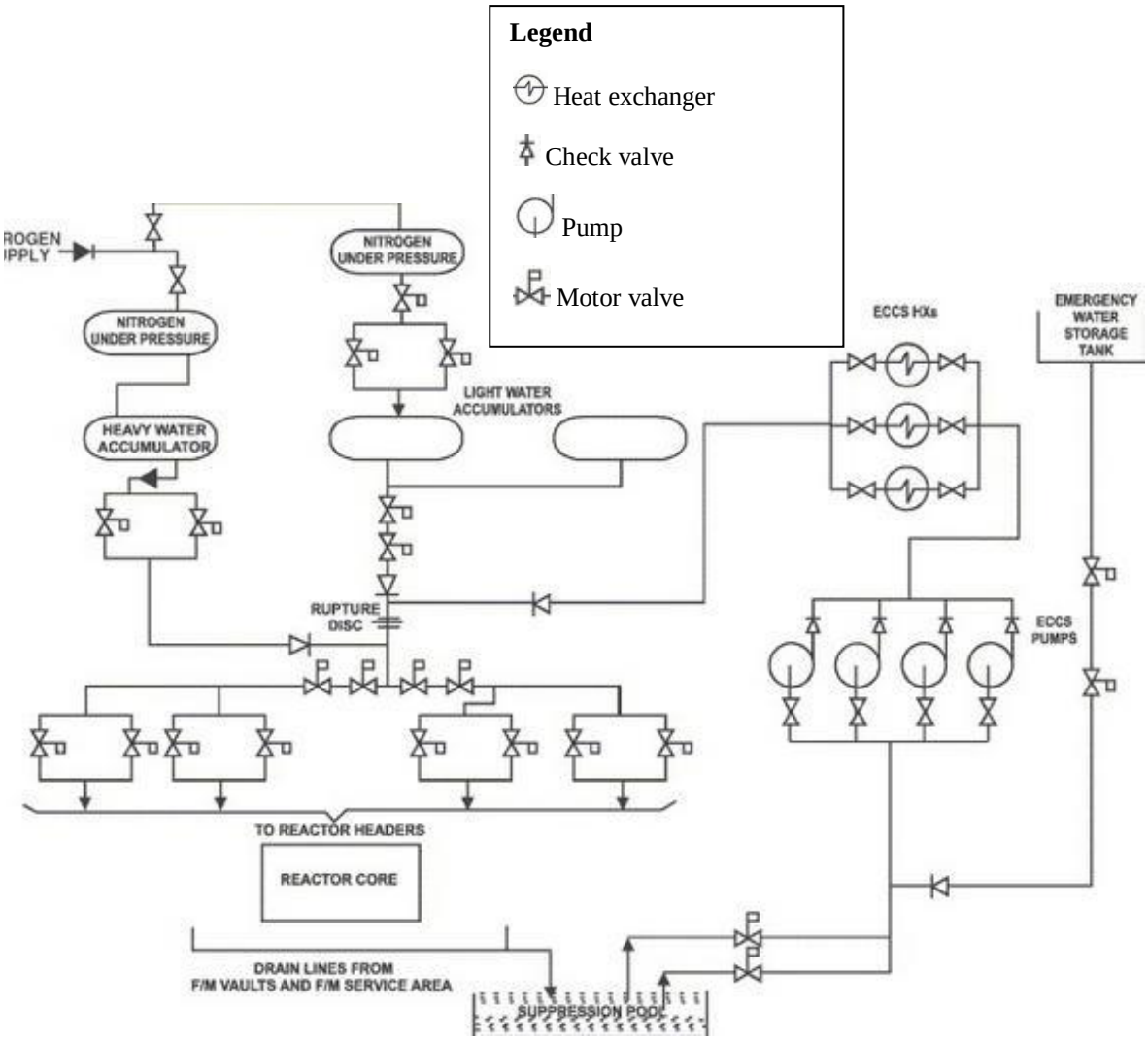

Figure 4.1: Mimic diagram of ECCS.

4.2.3 Control logic module

There are many control logics for the operation of ECCS equipments (valves, pumps, tanks, etc). For the illustration of our work, we describe the control logic for opening and closing a motorized valve based on Nitrogen tank pressure as follows:

1. On LOCA signal, the Level Transmitter senses the light water level in the light water accumulators.

2. If there is sufficient water, the two series valves will open.

The architecture of this control logic is depicted in figure 4.2. On LOCA condition, LT senses the level of the light water accumulator and passes on to the control logic, which compares the light water level with the threshold value. If it is greater than the threshold value, it de-energizes the relay to open the valve; else it will retain the relay in the energized (normal) state. Based on this logic, three important components are identified: Control logic, LT, Relay. Scenarios can be built, using UML, to refine and analyze the requirements of the stated control logic of valve actuation, for which HMSC are represented as Activity diagrams, shown in figure 4.3 and BMSC are represented as Sequence diagrams, shown in figure 4.4. Scenario specifications also provide solutions to the some important questions, related to reliability: i) which component is more significant for overall system reliability? ii) what is the impact of any component failure on the overall system reliability? iii) in what fashion the components should be arranged to obtain the predicted reliability? To answer these questions, we need to make reliability models at early phases of the SDLC.

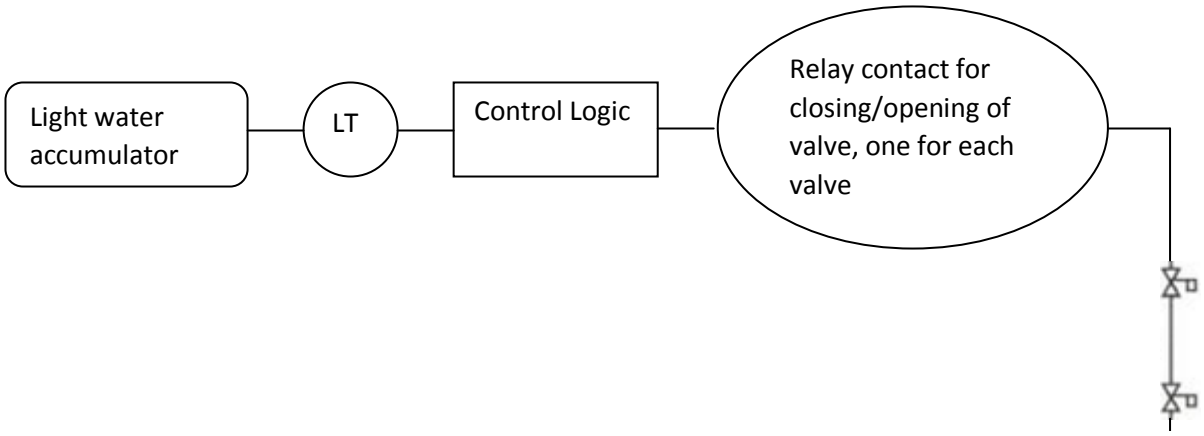

Figure 4.2: Architecture of valve control logic.

4.3 Defining attributes into scenario specifications

We introduce new attributes into scenario specifications of UML to demonstrate our approach for reliability prediction.

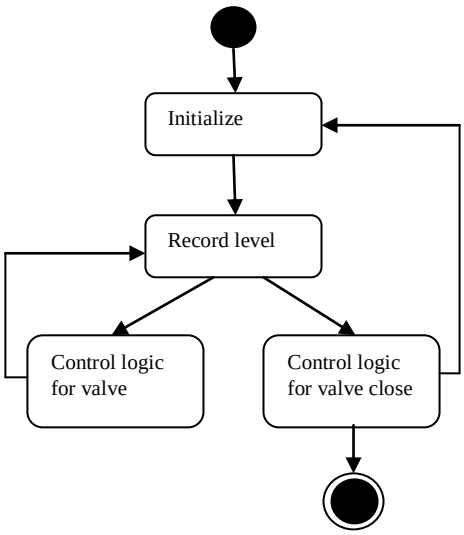

Figure 4.3: Activity diagram for valve actuation.

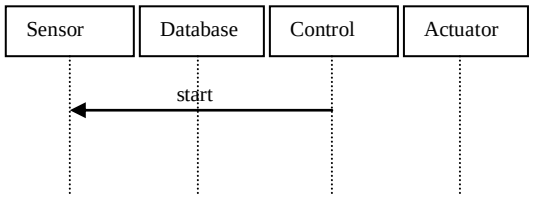

Figure 4.4(a) Sequence diagram for initialize

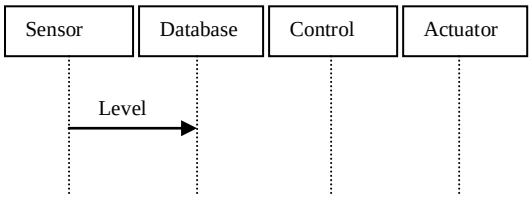

Figure 4.4(b) Sequence diagram for record level

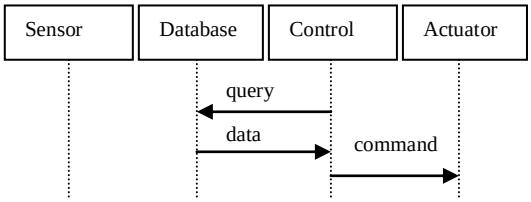

Figure 4.4(c) Sequence diagram for control logic for valve open

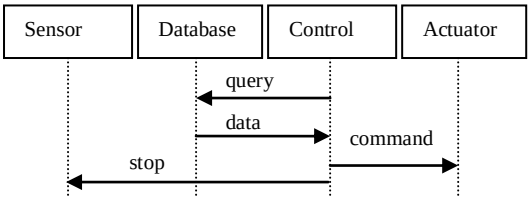

Figure 4.4(d) Sequence diagram for control logic for valve close

Figure 4.4: Sequence diagram for valve actuation.

4.3.1 Basic Message sequence Chart (BMSC)

It is a structure $b = (E, L, I, M, instance, label, order)$ where,

E: countable set of events that can be partitioned into a set of send and receive

events denoted by send (E) and receive (E), respectively.

L: set of message labels.

I: set of instance names.

M: $send(E) \rightarrow recieve(E)$, the pair of send(E) and receive(E), referred to as messages in M.

$instance$: $E \rightarrow I$ maps every event to the instance on which the event occurs. Given $i \in I$, the set $e \in E|instance(e) = i$ is denoted by i(E).

level: maps events to labels, requiring that $\forall(e, e') \in M$ where $label(e) = label(e')$ and if $(v, v') \in M$ and $label(e) = label(v)$ then $instance(e) = instance(v)$ and $instance(e') = instance(v')$.

order: set of total orders $\leq_i$ of an instance i, where $\leq_i$ corresponds to the top-down ordering of events on i.

The total ordering of events $\leq_i$ presented above model the relation where an event is considered to occur only after all the preceding events on the same instance, following a sequential representation of time.

4.3.2 High-level Message Sequence Charts (HMSC)

It is a graph of the form N, E, s_0, where N is a set of nodes, E is a set of edges connecting nodes, and $s_0 \in N$ is the initial node. A node n is adjacent to n' if $(n, n') \in E$.

4.3.3 Labeled Transition Systems (LTS)

It is a structure $P = (S, L, \Delta, q)$ where,

S: finite set of states.

L: $L = \alpha(P) \cup \tau$ where $\alpha(P)$ is a set of labels representing the communicating alphabet of P.

Δ: $\Delta \subseteq (S\pi, \epsilon \times L \times S)$ defines the labeled transitions between states, where no transition is originated from the states error, π,or end,ϵ.

q: $q \in S$ is the initial state.

4.4 A Technique for Early Software Reliability Prediction

System reliability in the early phases of SDLC can be obtained by transforming the scenario specifications into behavior models. We extend the approach given by Uchitel et al [27] by annotating the scenario specifications to explore reliability at the modeling level, thereafter analyzing the probability that the system can reach to a failure state. We illustrate our approach of computing reliability from scenario specifications followed by conversion into the representation of component behavior in LTS with probability weights. We consider the following assumptions in our approach:

1. The transition between the components follows Markov properties.

2. Failures are independent.

3. If a component C of reliability R_C wants to use the services of component C' of reliability R'_C, it will send a message to C'. Hence the reliability with which this service is performed is $R_C \times R'_C$. We also assume that the execution time to invoke the component is very short; hence is not included in the reliability computation.

In HMSC, there is only one initial and one final state. Multiple initial an final states can be defined with the concept of super-initial and super-final states [70].

To illustrate our approach, we re-define the BMSC by appending two more attributes $R_{ins} : I \rightarrow [0, 1]$, the reliability of an instance $i \in I$,and $P_e : E \rightarrow [0, 1]$, the probability that an event e is successful. Here $e \in recv(E)$ is the reliability of instance $i, R(i)$ and $instance(e) = i$. We also need to re-define HMSC by adding one more attribute $P_t : E \rightarrow [0, 1]$ to represent the transition probability between scenarios. Clearly,

$$\forall n \in N, \sum_{e \in S(n)} P_t(e) = 1,$$

$where \qquad S : N \rightarrow 2^E$

$S(n)$: the set of edges for the adjacent nodes of a node n and is defined as: $S(n) = e \in E | \exists n' \in N.e = (n, n')$

$P_t(i \rightarrow j)$: Transition probability from scenario i to j.

Our framework for reliability prediction consists of 4 phases, described below, as depicted in figure 4.5.

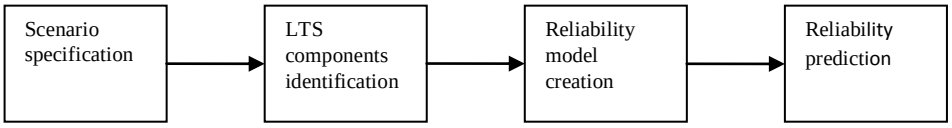

Figure 4.5: Framework for reliability prediction.

4.4.1 Phase1: Scenario Specification

In this phase we identify all the possible scenarios and probabilities of transition between them in our redefined HMSC. We also identify P_t and reliability of each component in our redefined BMSC. We reproduce the activity diagram, described in figure 4.3, which was used to represent HMSC, to represent the redefined HMSC in figure 4.6. Let,

R_S: reliability of Sensor component

R_D: reliability of Database component

R_C: reliability of Control component

R_A: reliability of Actuator component

The redefined BMSC would be same as the original BMSC as shown in figure 4.4, except for showing the components' reliabilities, shown in figure 4.7. Using the operational profile of 1 year for this control logic, we compute transition probabilities between

Table 4.2: Transition probabilities between activities of HMSC

$P_t(init \rightarrow rlevel)$	$P_t(rlevel \rightarrow vo)$	$P_t(rlevel \rightarrow vc)$	$P_t(vo \rightarrow rlevel)$	$P_t(vc \rightarrow init)$	$P_t(vc \rightarrow stop)$
1.0	0.8	0.2	0.2	0.3	0.7

Table 4.3: Reliabilities of components of BMSC

R_S	R_D	R_C	R_A
0.996	0.999	0.990	0.994

the components as given in table 4.2.

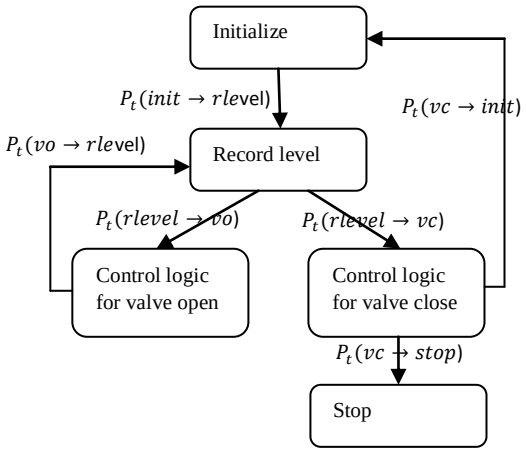

Figure 4.6: Modified Activity diagram for valve actuation with transition probabilities.

4.4.2 Phase2: LTS components identification

In this phase, we synthesize probabilistic-LTS [71] from the scenario specification that was shown in the first phase. The difference between LTS and probabilistic LTS is that LTS customizes the probabilistic-LTS by introducing a probability function to the labeled transitions between states, where the transition probability function P satisfies

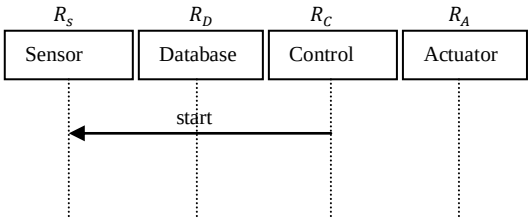

Figure 4.7(a) Sequence diagram for initialize

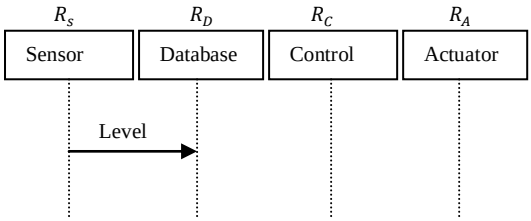

Figure 4.7(b) Sequence diagram for record level

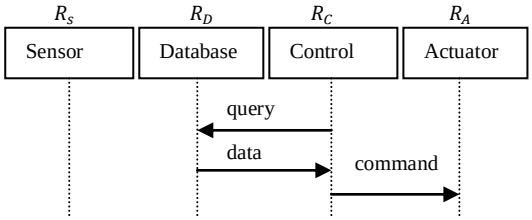

Figure 4.7(c) Sequence diagram for control logic for valve open

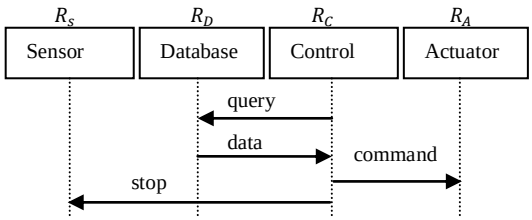

Figure 4.7(d) Sequence diagram for control logic for valve close

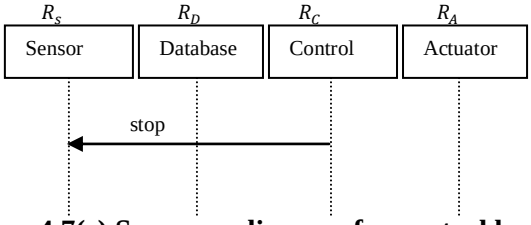

Figure 4.7(e) Sequence diagram for control logic for Stop

Figure 4.7: Modified Sequence diagram for valve actuation.

the following property:

$$\forall s \in S, \qquad \sum_{l \in L} \sum_{s' \in S} P(s, l, s') = 1$$

It is impractical to define an LTS with many states, for which Finite State Process (FSP) has been introduced [72-73]. For each component, the FSP process with weights on each action of that process is first synthesized. Each process represents the behavior of the corresponding component in the redefined BMSC with its weight. The algorithm to create a probabilistic-FSP from a scenario specification is given in Algorthim 1.

In the first step, the behavior of the component is built while traversing each re-defined BMSC. Finally in the second step, the FSP is built for the behavior of the

component defined by the redefined HMSC. The mapping from the redefined HMSC to the redefined BMSC is modeled and the possible adjacencies of nodes in the redefined HMSC, along with the function getadjacent() in figure 4.7, is modeled. In extension to Uchitel et al's approach, we map the adjacent nodes. In the getadjacent() function, the transition probabilities are mapped to the weights of the hidden transitions, termed internal transitions.

We present probabilistic LTS in figure 4.8 for this control logic. We apply the first and second steps to synthesize the redefined LTS. Each activity contains the redefined LTS from the created BMSC and so models the behaviour of the logic within that BMSC. The transitions between the redefined LTS, shown in the second step, correspond to the transitions between the BMSCs defined in the HMSC. Weights on the transitions are same as in the redefined HMSC, as shown in figure 4.6. It can be seen from the message 'data' in figure 4.7(c) and figure 4.7(d) that there can be "successful" transitions with probability R_{ctrl} and "unsuccessful" transitions with probability $1 - R_{ctrl}$. This action is only applicable to the transitions which are labelled with 'data' as in this case only some operations are being performed.

The weights should be computed properly to reduce each component probabilistic-LTS to first its minimal form followed by its deterministic form. We extend the approach given in [27] to introduce new assignments for accommodating the weights of the internal actions. The algorithm is given in Algorithm 2.

This algorithm eliminates the transitions, which merge the transition's target state with its source state, with the outgoing transitions of the target state becoming outgoing transitions of the source state. Because there can be multiple transitions from the initial source state, the weights of the transitions that are to be eliminated must be pushed to the new outgoing transitions, with the new weight on each such outgoing transition equal to its old weight times the weight of the eliminated transition.

There is a need to do filtration before running this algorithm. Remove all the transitions which lead to a stop state, terminate state or self-loop state. At the end, weights of the outgoing transitions of the resulting state may not be necessarily sum to one, hence

weights must be normalized. For determination of component probabilistic-LTS, we first find all the transitions which start from the same state and leave to different successor states, under the same action. Each such set of non-deterministic transitions is merged to have the same successor state, with the probability weights of the original transitions summed to form the weight for the transition to the merged successor state. Accounting for cycles in the state machine is required. The algorithm is given in Algorithm 3.

```
 pFSP(scenario s, component c){
∀ BMSC b∈s.getBMSC(),
getbehavior(c,b);
getadjacent(s.getHMSC().getinitnode(), s.getHMSC());
∀ ∈ s.getHMSC().getnodes()
        b=s.getHMSC().map(n);
CString node_behav=c.name()+b.name();
∀ ∈ s.getHMSC().getnodes()
       getadjacent(n,s.getHMSC());
cout«″c.name()″«″intaction″;
CString getbehavior(component c, BMSC b){
CString s, wt, err, mesg;
int next;
double rel, error;
instance inst=b.getinstance(c);
CString label=c.name()+b.name();
for(int i=0; i<inst.size(); i++){
   next=i+1;
   mesg=i.getEvent(a).label();
   rel=i.getEvent(a).weight();
if rel<1.0 then
   error=1-rel;
end
   s=s+wt+err;
   err=mesg;
   wt=label+rel+label+next;
   s=s+wt+err;
}
s=s+"stop";
return s;
}
CString getadjacent(node n,HMSC h){
```

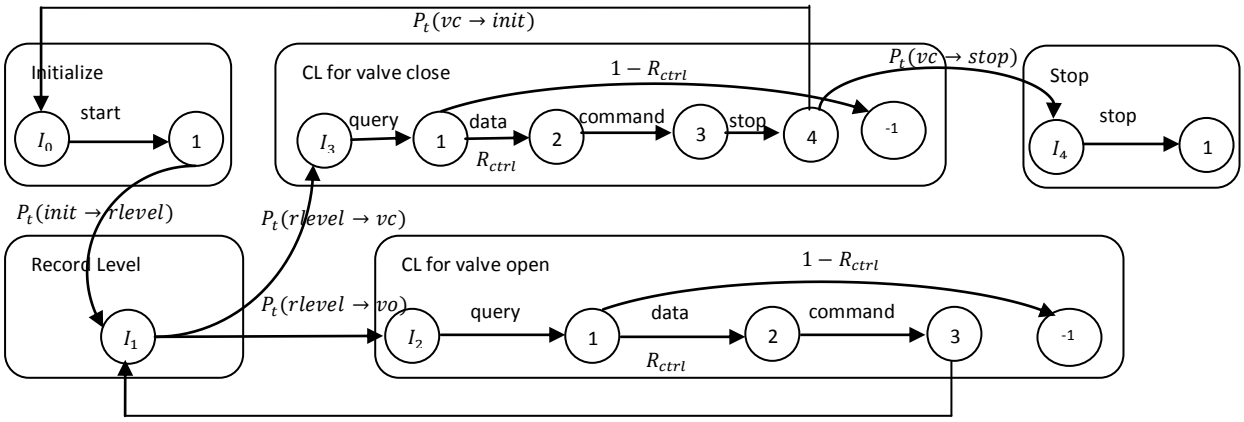

Figure 4.8: Probabilistic-LTS for control logic.

Input $I = (S, L, \Delta, q)$;

Output $O = (S', L', \Delta', q')$;

$O \leftarrow I \setminus \{p_t\}$;

$q' \leftarrow q$;

$\Delta' \leftarrow \phi$;

for each $s \in S$ do

  $\Delta' \leftarrow \Delta' \cup s, p, l, s$, where $l \neq p_t$;

  for each $n \in Path(s)$ do

    $\Delta' \leftarrow \Delta' \cup \{(s, p \times p', l, u) : (n_i, p', l, u) \in \Delta, l \neq p_t$, for $0 \leq i < |n|\}$;

  end for

end for

where:

$Path(s)$ is a sequence of states $n = s_0, s_1, \ldots$;

if $s_0 == s$ **then**

  $s \in S$ where $s_{i+1} \in adj(s_i)$;

  if $(s, p, l, s') \in \Delta$ **then**

    $s' \in adj(s)$ where $p_t \in L$;

  end

end

Algorithm 2: Algorithm to minimize probabilistic-LTS

Input $I = (S, L, \Delta, q)$;

Output $O = (S', L', \Delta', q')$;

$\Delta' \leftarrow \phi$;

$q' \leftarrow q$;

for each $s \in S$ do

  for each $s' \in adj(s), (s, p, l, s') \in \Delta$ do

    $adj'(s) \leftarrow adj(s)$;

if $\exists s'' \in adj'(s), (s, p', l', s'') \in \Delta, l = l'$ **then**

  for each $s' \in adj(s'), s'' \neq s, (s', p'', l'', s'') \in \Delta$ do

  $\Delta' \leftarrow \Delta' \cup (s', p \times p'', l'', s'') : (s'', p'', l'', i) \in \Delta$;

  end for

  for each $s'' \in adj(s), s' \neq s'', (s, p', l', s'') \in \Delta$ do

  if $l = l'$ **then**

    for each $s''' \in adj(s''), s''' \neq s, (s'', p'', l'', s''') \in \Delta$ do

    $\Delta' \leftarrow \Delta' \cup (s', p' \times p'', l'', s''') : (s''', p'', l'', u) \in \Delta$;

    end for

    $p \leftarrow p + p'$;

  end

  $\Delta' \leftarrow \Delta' \cup \{s, p, l, s'\}$;

  $adj'(s) \leftarrow adj'(s) - \{s''\}$;

  end for

end

 end for

end for

Algorithm 3: Identification of probabilistic-LTS

4.4.3 Phase3: Reliability model creation

In the third phase we create the reliability model. Using the algorithm, the minimal LTS for our control logic can be derived from table 4.2 and table 4.3, as shown in figure 4.9. From figure 4.9 we can see that the sum of the outgoing transitions from each state

in probabilistic-LTS is 1. Here state E represents correct termination and the state -1 represents fault termination. Figure 4.9 represents the reliability model that extracts the system reliability, which resembles Discrete Time Markov Chain.

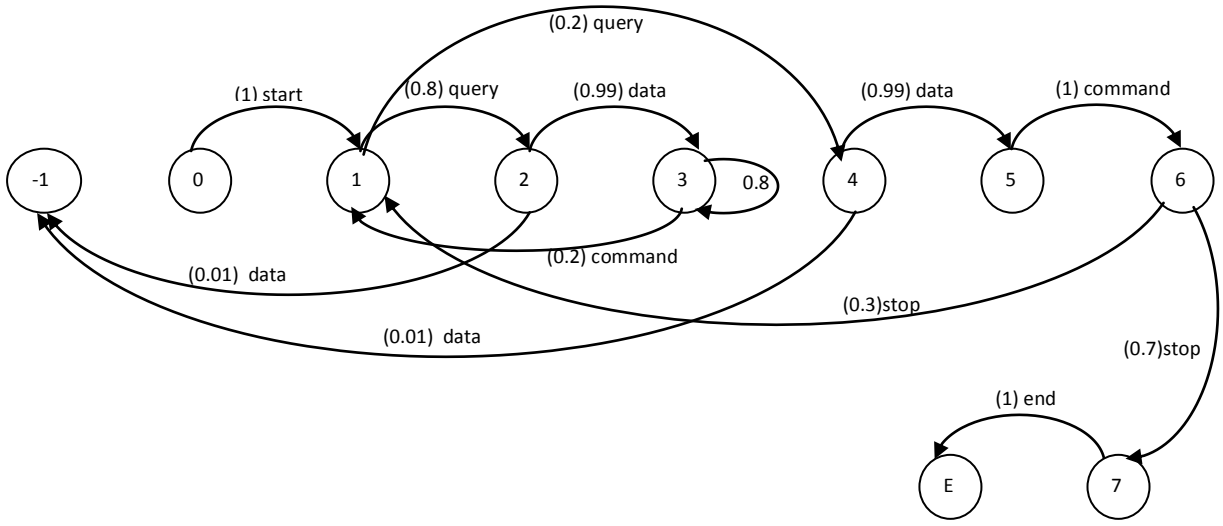

Figure 4.9: probabilistic-LTS for control logic with transitions.

4.4.4 Phase4: Reliability prediction

This is the final phase of our framework which was illustrated in figure 4.5. In the previous phase we create a reliability model, which is DTMC. Cheung et al [18] proposed a framework to predict the reliability of software components, the final phase of which we use here to predict the reliability of our control logic. Let the representation of all the states in this DTMC model be given by equation 4.1. The transition probability matrix from the created DTMC model is given in equation 4.2.

$$\{0, 1, 2, 3, 4, 5, 6, 7, E, -1\} = \{S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, F\} \tag{4.1}$$

where $S_i \in$ behavioural state $\forall i = 0 \rightarrow 8$ and F is fault state.

$$
P = \begin{array}{c} \\ S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \\ S_7 \\ F \\ S_8 \end{array}
\begin{array}{c}
\begin{array}{cccccccccc} S_0 & S_1 & S_2 & S_3 & S_4 & S_5 & S_6 & S_7 & F & S_8 \end{array} \\
\begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.8 & 0 & 0.2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.99 & 0 & 0 & 0 & 0 & 0.01 & 0 \\
0 & 0.2 & 0 & 0.8 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0.99 & 0 & 0 & 0.01 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0.3 & 0 & 1 & 0 & 0 & 0 & 0.7 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.4 & 0.5 & 0.2 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{array}
\tag{4.2}
$$

Let $p_i(t)$ be the probability that a component is in state i at time t. When component executes for a very long time ($t \to \infty$), these probability converges and leads to stationary distribution [18].

$$
\overrightarrow{p} = [p(S_0), p(S_1), p(S_2), p(S_3), p(S_4), p(S_5), p(S_6), p(S_7), p(F), p(S_8)] \tag{4.3}
$$

Also,

$$
\sum_{i \in M} p(i) = 1 \tag{4.4}
$$

$$
\overrightarrow{p} = \overrightarrow{p} P \tag{4.5}
$$

These are linear equations and can be solved by standard numerical techniques [41]. Hence the reliability of the communication module can be computed by removing the probability of unsuccessful packet sent, which is denoted by F, i.e.

$$
R = 1 - p(F) \tag{4.6}
$$

So from equation 4.5, we get equation 4.7

$$
\begin{aligned}
&[S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7, F, S_8] \\
&= [S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7, F, S_8]
\begin{array}{c}
\\ S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \\ S_7 \\ F \\ S_8
\end{array}
\begin{array}{c}
\begin{array}{cccccccccc} S_0 & S_1 & S_2 & S_3 & S_4 & S_5 & S_6 & S_7 & F & S_8 \end{array} \\
\begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.8 & 0 & 0.2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.99 & 0 & 0 & 0 & 0 & 0.01 & 0 \\
0 & 0.2 & 0 & 0.8 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0.99 & 0 & 0 & 0.01 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0.3 & 0 & 1 & 0 & 0 & 0 & 0.7 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.4 & 0.5 & 0.2 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{array}
\end{aligned}
\tag{4.7}
$$

Solving equation 4.7 we get the following linear equations

$$\begin{aligned} S_0 &= 0, \\ S_1 &= S_0 + 0.2S_3 + 0.3S_6, \\ S_2 &= 0.8S_1, \\ S_3 &= 0.99S_2 + 0.8S_3, \\ S_4 &= 0.2S_1, \\ S_5 &= 0.99S_4, \\ S_6 &= S_5, \\ S_7 &= 0.7S_6, \\ F &= 0.01S_2 + 0.01S_4, \\ S_8 &= S_7 \end{aligned} \tag{4.8}$$

Using equation 4.4, we get

$$S_0 + S_1 + S_2 + S_3 + S_4 + S_5 + S_6 + S_7 + F + S_8 = 1 \tag{4.9}$$

Solving equations 4.8 & 4.9, we get

$$S_0 = 0; S_1 = 0.151; S_2 = 0.1208; S_3 = 0.56; S_4 = 0.0302;$$
$$S_5 = 0.03; S_6 = 0.03; S_7 = 0.021; S_8 = 0.021; F = 0.00151$$

Hence the reliability of the control logic, which we take as a case study, given by equation 4.6 is

$$R = 1 - p(F) = 1 - 0.00151 = 0.99849 \tag{4.10}$$

4.5 Sensitivity Analysis

We carry out analysis to determine the impact of change in components' reliabilities and change in the transition probabilities in probabilistic-LTS on the system reliability.

Determining the impact of changes in component reliabilities of the system reliability helps to identify the components that are the most critical for system reliability. This information will help the system designer or architect to design these critical components carefully, by incorporating redundancy, etc. We change the reliability of one component at a time and see the impact on the system reliability. The reliability of the system architecture as a function of components' reliability is shown in figure 4.10.

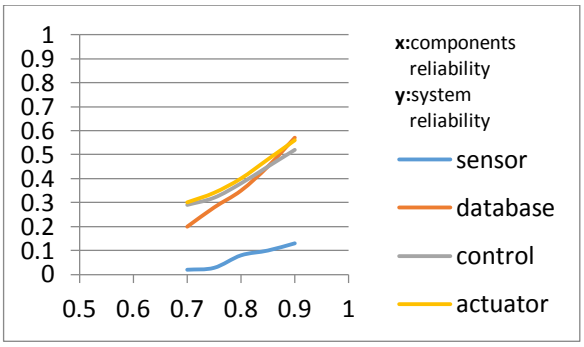

Figure 4.10: System Reliability as a function of components reliability.

It can be noticed that as the sensor's reliability decreases, the system reliability also decreases rapidly. Hence the impact of the sensor's reliability on the system reliability is very high, and so the sensor is a very critical component.

We consider this result because all the valve operations are based on the sensor (LT) measurement. If LT gives the wrong measurement of the tank level, the system will behave in an unspecified manner, which can jeopardize the safety.

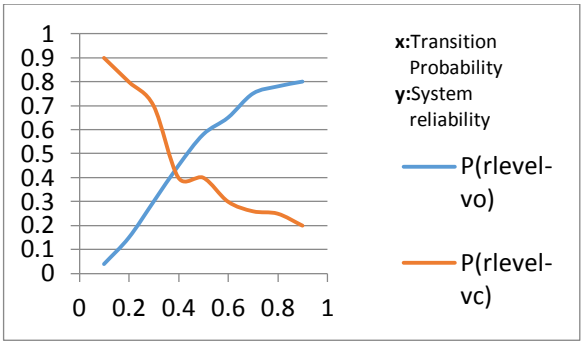

Figure 4.11: System Reliability as a function of transition reliability.

We also analyze the sensitivity of the system reliability as a function of the scenario

transition probabilities. We know that sum of outgoing transitions from a scenario is unity. If we change the value of $P_t(rlevel \rightarrow vo)$ from 0.8 to 0.1 and the value of $P_t(rlevel \rightarrow vc)$ from 0.2 to 0.9, we get the results as shown in figure 4.11. We can see that as the transition probability $P_t(rlevel \rightarrow vo)$ decreases, system reliability also decreases and as the transition probability $P_t(rlevel \rightarrow vo)$ increases, system reliability decreases.

4.6 Experimental Validation

To validate the correctness of our approach and the accuracy of our results, we computed the reliability of the same logic using the operational profile of one year. We developed a CBS, known as Test Facility, which is responsible to ensure the healthiness of all the ECCS equipments, logics and interlocks. Test Facility is used to monitor the ECCS process parameters round the clock and keeping in view of ECCS target reliability of 10^3 years/year , it tests the ECCS equipment once in a month. Test Facility has a feature to log every action of the operator, every event and every changed state of any equipment or process parameters. During the testing, the conditions or logics are simulated and equipment operations, relevant to those conditions or logics are monitored and logged. For the control logic that we have taken as a case study, we took the log of the valve operation, as given in table 4.4. As per the specification of the system, the total number of times the valve should open is 7 and total number of times the valve should close is also 7. Let

t =number of months.

n_o =number of times the valve has opened.

n_c =number of times the valve has closed.

n_{fo} =number of times the valve has failed to open.

n_{fc} =number of times the valve has failed to close.

$n_f = n_{fo} + n_{fc}$ =number of times the valve has failed.

n =total number of times the valve should succeed.

R_t =Reliability of valve operation logic up to time t.

Therefore failure probability if given by equation 4.11,

$$Failure\ probability = \frac{n_f}{n} \tag{4.11}$$

and hence Reliability is given by equation 4.12,

$$Reliability = 1 - Failure\ probability \tag{4.12}$$

Considering data of 1 month from table4,

$$\begin{aligned} n_f &= 0, \\ n &= 14 \\ \therefore Failure\ probability &= \frac{0}{14} = 0 \\ Reliability\ R_1 &= 1 - Failure\ probability \\ &= 1 - 0 = 1 \end{aligned}$$

Similarly the reliability for the rest of the period can be computed, which is shown in table 4.4.

From table 4.4, it can be noticed that in the 8th month, the valve failed to open only 1 time, and thereafter till 12 months, no failure has been observed. So the reliability of the valve operation logic that has been computed is 0.994. We could only get the data of one year, as the new version was installed at the Nuclear Power Plant one year back. So we can write,

$$R_{12} = 0.994 \tag{4.13}$$

If we compare the reliability figure from the operation profile of 1 year that is 0.994, with the reliability figure that we predicted through system modeling that is 0.99849, we notice that both the figures are almost same, which is an admirable result.

Table 4.4: Operational profile of 1 year for valve operation logic

t	n_0	n_c	n_{fo}	n_{fc}	n_f	n	R_t
1	7	7	0	0	0	14	1
2	14	14	0	0	0	28	1
3	21	21	0	0	0	42	1
4	28	28	0	0	0	56	1
5	35	35	0	0	0	70	1
6	42	42	0	0	0	84	1
7	49	49	0	0	0	98	1
8	55	56	1	0	1	111	0.991
9	62	63	1	0	1	125	0.992
10	69	70	1	0	1	139	0.993
11	76	77	1	0	1	153	0.994
12	83	84	1	0	1	167	0.994

4.7 Conclusion

We have proposed an approach to predict software system reliability as a function of its components' reliabilities. The methodology consists of specifying the scenarios with probabilistic properties along with the creation of probabilistic-LTS from the scenario specifications. This probabilistic-LTS helps to derive the software system reliability model. The process has been illustrated in detail with the help of a case study of a Nuclear Power Plant system. We have also given the sensitivity analysis, in which we have analyzed the impact of components' reliabilities on the software system reliability. This impact analysis helps to find the criticality of the components of the system. Through this information, the system architect can take the preventive action. Further, the impact of transition probabilities on the software system reliability has been shown. Finally we have validated our approach by comparing the predicted reliability from our approach and from the operational profile data of 1 year.